Exercises: Data Representation and Manipulation

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

1. Reverse Array

Write a program that **reverses** and **prints** an array. Use **recursion**.

Examples

Input	Output
1 2 3 4 5 6	6 5 4 3 2 1

2. Nested Loops To Recursion

Write a program that simulates the execution of **n** nested loops **from 1 to n** which prints the values of all its iteration variables at any given time on a single line. **Use recursion.**

Examples

Input	Output	Solution with nested loops (assuming n is positive)

2	1 1
	1 2
	2 1
	2 2
		`int n = 2;`
		
		`for (int i1 = 1; i1 <= n; i1++){`
		`  for (int i2 = 1; i2 <= n; i2++){`
		`  System.out.println(i1 + " " + i2);`
		`  }`
		`}`

3	1 1 1
	1 1 2
	1 1 3
	1 2 1
	1 2 2
	…
	3 2 3
	3 3 1
	3 3 2
	3 3 3
		`int n = 3;`
		
		`for (int i1 = 1; i1 <= n; i1++){`
		`  for (int i2 = 1; i2 <= n; i2++){`
		`  for (int i3 = 1; i3 <= n; i3++){`
		`  System.out.println(i1 + " " + i2 + " " + i3);`
		`  }`
		`  }`
		`}`

3. * Implement Binary Search using Recursion

The **Binary Search** algorithm also can be implemented using **recursion**. Knowing how recursion works try implementing it on your own.

Read a sequence of **sorted** numbers on the first line and a single number on the second from the console. **Find the index** of the number in the given array. Return **-1** if the element is **not present** in the array.

Examples

Input	Output
1 2 3 4 5	0

1	
1 2 3 4 5 6	-1

Hints:

Start by defining a class with a **method**:

```
public class BinarySearch {
    public static void main(String[] args) throws IOException {
        //TODO: read the elements from the console

        System.out.println(binarySearch(elements, key, lo: 0, elements.length));
    }

    private static int binarySearch(int[] nums, int key, int lo, int hi) {

    }
}
```

Inside the method, **check** if the **lower bound is smaller than or equal the higher bound**:

```
if (lo <= hi) {
    //TODO: Find index of key
}

return -1;
```

Inside the **if-block**, we need to find the **midpoint**:

```
int mid = lo + (hi - lo) / 2;
```

If the key is to the **left** of the midpoint, continue searching in the **left half** of the collection. If the key is to the **right** of the midpoint, continue searching in the **right half**:

```
int mid = lo + (hi - lo) / 2;
if (key < nums.get(mid)) {
    return binarySearch(nums, key, lo: lo, hi: mid);

} else if (key > nums.get(mid)) {
    return binarySearch(nums, key, lo: mid + 1, hi: hi);

} else {
    return mid;
}
```

That's it! Good job!

4. Combinations Count

Do you remember the Pascal's triangle we built in one previous lection? The values that the triangle holds are the so called binomial coefficients. Binomial coefficients (read as "*n* choose *k*") give us the number of ways we can choose *k* elements from a set of *n* elements and are calculated using the following formula:

$$C_n^k = \binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

Given a **n** and **k**, calculate the number of possible **n choose k** combinations. Use **Recursion.**

Examples

Input	Output
3 2	3
49 6	13983816

5. Chocolates

You are given an array of **n** integers where each value represents number of chocolates in a packet. Each packet can have variable number of chocolates. There are **m** students. Your task is to distribute chocolate packets such that:

1. Each student gets one packet.
2. The difference between the number of chocolates in packet with maximum chocolates and packet with minimum chocolates given to the students is minimum.

On the first line you will get the number of packets **n.** On the next line you will get all packets, separated by a space and a comma. On the last line you will get the number of students **m.**

Input	Output	Comments
7 7, 3, 2, 4, 9, 12, 56 3	Min Difference is 2.	We have 7 packets of chocolates and we need to pick 3 packets If we pick 2, 3 and 4, we get the minimum difference between maximum and minimum packet sizes.
8 3, 4, 1, 9, 56, 7, 9, 12 5	Min Difference is 6.	We pick 3,4,7,9,9 and the output is 9-3 = 6

6. Overlapping Intervals

An interval is represented as a combination of start time and end time. Given a set of **n** intervals, check if any two intervals overlap.

On the first line you will get the number of packets **n.** On the next line you will get each interval. Its start and end time will be separated by a comma.
Print "**true**" if any two intervals overlap, otherwise print "**false**".

Input	Output	Comments
4 1,3 5,7 2,4 6,8	true	The intervals {1,3} and {2,4} overlap.
4 1,3 7,9 4,6 10,13	false	No pair of intervals overlap.

Hints:

- You can decrease the complexity of your algorithm by sorting the intervals first.

7. Find the Missing Number

You are given a list of **n-1** integers and these integers are in the range of **1 to n**. There are no duplicates in the list. One of the integers is **missing** in the list. Find the missing integer.

You will get an integer **n** on the first line and **n-1** integers, separated by a space and a comma, on the second line;

Input	Output
8 1, 2, 4, 6, 3, 7, 8	5
12 3, 4, 1, 11, 2, 5, 7, 9, 12, 8, 10	6

8. Multiply Big Number

You are given two lines - the first one can be a really big number (0 to 10^{50}). The second one will be a single digit number (0 to 9). You must display the product of these numbers.

Note: do not use the **BigInteger** or **BigDecimal** classes for solving this problem.

Examples

Input	Output	Input	Output
23 2	46	9999 9	89991

Input	Output
923847238931983192462832102 4	369538895572793276985132840 8

9. *Phone Numbers

You are given a string, holding ASCII characters. Find all **name -> phone number** pairs, format them and print them in an **ordered list** as list items.

The **name** should be at least **one letter long, can contain only letters** and should **always start with an uppercase letter**.

The **phone number** should be at least **two digits long**, **should start and end with a digit** (**optionally**, there might be a **"+" in front)** and might **contain** the following symbols in it: **"(", ")", "/", ".", "-", " "** (left and right bracket, slash, dot, dash and whitespace).

Between a name and the phone number there might be **any number of symbols, other than letters and "+"**.

Between the name -> phone number pairs there might be **any number of ASCII symbols**.

The output format should be **[name]: [phone number]** (there is **one space between** the name and the phone number). Clear any characters other than digits and "+" from the number when printing the output.

Input

The input will be read from the console. It will consist of several lines holding the input string. The command "**END**" denotes the end of input.

Output

The output should hold the **resulting ordered list (on a single line)**, or a single **paragraph,** holding "**No matches!**"

Constraints

- The **numbers string** will hold only **ASCII** characters (no Unicode).
- Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input
Angel$(*^#**029661234**!@#**Pesho** ,.' **+3592/9653241**;'":{},. **Ivan 0888 123 456 John**-=_**555.123.4567** Stoian!@#$#@ Gosho)=_* **Steven** #$(*&**+1-(800)-555-2468** END
Output (li items are separated on new lines for convenience)
- Angel: 029661234   - Pesho: +35929653241   - Ivan: 0888123456   - John: 5551234567   - Steven: +18005552468

Input
There are no phone numbers here!!! END
Output
No matches!

10. **Semantic HTML

You are given an **HTML code**, written in the old **non-semantic** style using tags like **`<div id="header">`**, **`<div class="section">`**, etc. Your task is to write a program that **converts this HTML to semantic HTML** by changing tags like **`<div id="header">`** to their semantic equivalent like **`<header>`**.

The non-semantic tags that should be converted are **always `<div>`**s and have either **`id`** or **`class`** with one of the following values: "**`main`**", "**`header`**", "**`nav`**", "**`article`**", "**`section`**", "**`aside`**" or "**`footer`**". Their corresponding closing tags are always followed by a comment like **`<!-- header -->`**, **`<!-- nav -->`**, etc. staying at the same line, after the tag.

Input

The input will be read from the console. It will contain a variable number of lines and will end with the keyword "**END**".

Output

The output is the semantic version of the input HTML. In all converted tags you should **replace multiple spaces** (like **`<header  style="color:red">`**) with a single space and remove excessive spaces at the end (like **`<footer  >`**). See the examples.

Constraints

- Each line from the input holds either an HTML **opening tag** or an HTML **closing tag** or HTML **text content**.
- There will be no tags that span several lines and no lines that hold multiple tags.
- Attributes values will always be enclosed in **double quotes "**.
- Tags will never have **id** and **class** at the same time.
- The HTML will be **valid**. Opening and closing tags will match correctly.
- **Whitespace** may occur between attribute names, values and around comments (see the examples).
- Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output

`<div id="header">`
`</div> <!-- header -->`
`END`
	`<header>`
	`</header>`

`<div style="color:red" id="header">`
`</div> <!-- header -->`
`END`
	`<header style="color:red">`
	`</header>`

`<div class="header" style="color:blue">`
`</div> <!-- header -->`
`END`
	`<header style="color:blue">`
	`</header>`

`<div align="left" id="nav" style="color:blue">`
`  <ul class="header">`
`  <li id="main">`
`  Hi, I have id="main".`
`  </li>`
`  </ul>`
`</div> <!-- nav -->`
`END`
	`<nav align="left" style="color:blue">`
	`  <ul class="header">`
	`  <li id="main">`
	`  Hi, I have id="main".`
	`  </li>`
	`  </ul>`
	`</nav>`

`  <p class = "section" >`
`  <div style = "border: 1px" id = "header" >`
`  Header`
	`  <p class = "section" >`
	`  <header style = "border: 1px">`
	`  Header`

```
        <div id = "nav" >
            Nav
        </div>   <!--   nav   -->
    </div>  <!--header-->
  </p> <!-- end paragraph section -->
END
```

```
        <nav>
            Nav
        </nav>
    </header>
  </p> <!-- end paragraph section -->
```